L4: Entry 3 of 9 File: USPT Dec 7, 1999

DOCUMENT-IDENTIFIER: US 5999931 A
TITLE: Concurrency control protocols for management of replicated data items in a distributed database system

Brief Summary Text (4):
A distributed database system typically includes a number of physical sites connected by a network. The physical sites may be, for example, centralized database systems such as data warehouses or data marts, or remote customer sites such as automatic teller machines or desktop personal computers. Many database systems support data processing transactions at multiple user sites. For example, a transaction operating at a particular user site may access a primary copy of a data item or record of a central database, while transactions at other user sites utilize replicated versions of the data item. A significant problem arises when the transactions at the different user sites attempt to update different replicated versions of the same data item, which may result in inconsistent replicated versions of a given data item. The problem of concurrent access to consistent replicated data has become increasingly challenging with the advent of large-scale distributed data warehouses and data marts, and the increasing use of distributed data in often-disconnected mobile computers. For example, data warehouses or data marts are now typically configured with storage capacities on the order of 0.5 to 3 terabytes, with approximately 10% to 25% of the stored data items being used in a replicated form, and about 10% of the data items being updated each day. These systems may require that all data item updates be reflected within a relatively short time period, such as 10 minutes or less. Other features of distributed database systems are described in, for example, A. A. Helal, A. A. Heddaya and B. B. Bhargava, "Replication Techniques in Distributed Systems," Kluwer Academic Publishers, 1996; C. Pu and A. Leff, "Replica Control in Distributed Systems: an Asynchronous Approach," Proceedings of ACM-SIGMOD 1991 International Conference on Management of Data, Denver, Colo., pp.377-386, May 1991; and J. Sidell, P. M. Aoki, S. Barr, A. Sah, C. Staelin, M. Stonebraker and A. Yu, "Data Replication in Mariposa," Proceedings of the Twelfth International Conference on Data Engineering, New Orleans, La., 1996, all of which are incorporated by reference herein.

Brief Summary Text (6):
Other known update techniques, generally referred to as "lazy" propagation techniques, address the above-described update propagation problem. Under lazy propagation, only one replica of a particular data item is updated by a transaction utilizing that data item. A separate transaction runs on behalf of the original transaction at each site at which update propagation is required. Lazy propagation effectively reduces transaction size but creates the possibility of two or more transactions committing conflicting updates to a data item if the transactions operate on different replicas. For example, a transaction T.sub.1 could update a data item d using the replica at a site s.sub.1 while a transaction T.sub.2 updates the replica of d at another site s.sub.2. If both transactions T.sub.1 and T.sub.2 commit an update of their replicas of the data item, the distributed system discovers the conflict only when the updates are propagated. Such conflicts may require either update reconciliation or the use of compensating transactions, as described in H. F. Korth, E. Levy and A. Silberschatz, "A Formal Approach to Recovery by Compensating Transactions," Proceedings of the Sixteenth International Conference on Very Large Databases, Brisbane, Australia, pp. 95-106, August, 1990, which is incorporated by reference herein.

Brief Summary Text (8):
Conventional lazy propagation techniques may also cause an update transaction to read "old" replicas of some data items, resulting in an execution that generates an inconsistent database

state. The problem may be alleviated to some extent by augmenting the above-noted lazy-master approach with restrictions on how the primary copies of data items are selected, as described in P. Chundi, D. J. Rosenkrantz and S. S. Ravi, "Deferred Updates and Data Placement in Distributed Databases," Proceedings of the Twelfth International Conference on Data Engineering, New Orleans, La., 1996, which is incorporated by reference herein. However, the resulting update propagation remains unduly susceptible to deadlocks and therefore is unsuitable for use in applications such as large-scale distributed systems with mobile computers.

Brief Summary Text (12):
A first illustrative concurrency-control technique in accordance with the invention is referred to as a static global serializability (SGS) protocol. As part of this protocol, a given transaction T.sub.i submits its set of data items in a preprocessing step, the transaction is assigned a timestamp, and the set is broadcast to all physical sites in the system. The origination site of a given transaction T.sub.i determines whether a cycle will be created in the replication graph if T.sub.i is allowed to proceed. If T.sub.i will not create a cycle in the graph, it is allowed to proceed at its origination site. When T.sub.i submits a write operation on a secondary copy of a data item, the timestamp of T.sub.i is compared to that of the transaction which executed the last write operation the same data item. If the T.sub.i timestamp is less than that of the last write on the data item, the T.sub.i write operation is not performed. Otherwise, the T.sub.i write operation is sent to the local database management system for execution. Transaction T.sub.i is removed from the replication graph when T.sub.i enters the above-described completed state. The SGS protocol thus applies a technique known as the Thomas Write Rule (TWR) to write-write conflicts on secondary copies of data items, and thereby achieves substantially reduced communication overhead.

Brief Summary Text (13):
A second illustrative concurrency-control technique in accordance with the invention is referred to as a commit-oriented protocol (COP). This protocol defers testing of a replication graph or other suitable globally consistent representation until a transaction is ready to commit. Each transaction is allowed to proceed at its origination site independently of other transactions that are executing at other sites. The only coordination required is when a given transaction T.sub.i submits the commit operation at its origination site. In this case, the transaction T.sub.i and its virtual sites are tentatively included in the replication graph, and a determination is made as to whether the resulting graph is acyclic. If the resulting graph is acyclic, the tentative changes are made permanent, and the commit operation is performed. Otherwise, the transaction T.sub.i is aborted. In this protocol, none of the transactions is permitted to wait, and therefore no distributed deadlocks can occur during transaction processing. The above-noted illustrative SGS and COP protocols guarantee global serializability and freedom from distributed deadlock without relying on any particular properties of the local database management systems running at the physical sites. In comparison to prior protocols, the new protocols reduce the communication required to coordinate transactions by approximately a factor of r, where r is the average number of operations per transaction.

Detailed Description Text (2):
The present invention will be illustrated below in conjunction with exemplary distributed database configurations. It should be understood, however, that the invention is not limited to use with any particular type of database or database configuration, but is instead more generally applicable to any database application in which it is desirable to provide consistency and concurrency across multiple replicated versions of data items. The term "data item" as used herein refers to a stored data element, a set of stored data elements, a record or any other type of data stored in a database or other storage system. The term "physical site" is intended to include remote customer sites, central databases, portable computers or other mobile computing devices, as well as any other physical location at which a data item may be processed. A "virtual site" refers generally to a set of data items accessed by a particular transaction at a particular physical site. A given virtual site VS.sub.i.sup.j includes a set of data items S.sub.i.sup.j which are accessed by a transaction T.sub.i at a physical site s.sub.j. The term "transaction" as used herein refers to any sequence of data processing operations which involves reading, writing or otherwise utilizing a data item or items, and

which can be terminated by either a commit or an abort operation. The term "committed" refers to the state of a transaction which has executed all of its operations which were to be performed at its origination site. The term "replication graph" refers to an exemplary representation of relationships between transactions and virtual sites, and includes a first set of identifiers for at least a subset of the transactions and a second set of identifiers for at least a subset of the virtual sites. The term "distributed database system" should be understood to refer to any type of system in which data items are stored or processed in multiple physical sites. As noted above, a schedule of transaction operations is "serializable" if running the transactions concurrently in accordance with the schedule yields the same results as running the transactions in some sequential order. Global serializability refers to serializability as applied to execution of all transactions in a system regardless of the physical sites at which particular transactions operate.

Detailed Description Text (3):
FIG. 1 shows an exemplary distributed database system 10 in which the present invention may be implemented. The system 10 includes N physical sites 12-i, i=1, 2, . . . N connected by a network 14. The network 14 may be a local area network, a metropolitan area network, a wide area network, a global data communications network such as the Internet, a private "intranet" or "extranet" network or any other suitable data communication medium. The physical sites 12-i may be centralized database systems such as data warehouses or data marts, remote customer sites such as automatic teller machines or desktop personal computers, portable computers or other mobile computing devices, or any other type of data processors. The operation of mobile computing devices in a distributed database system is described in greater detail in H. F. Korth and T. I. Imielinski, "Introduction to Mobile Computing," in Mobile Computing, pp. 1-39, Kluwer Academic Publishers, 1996, which is incorporated by reference herein. Certain of the interconnections between the various elements of the system 10 may be via relatively low-speed telephone line modem connections, such that network bandwidth is a scarce resource and the round-trip time for a message and acknowledgment is relatively large. Each of the N physical sites 12-i includes a processor 18-i and a database 20-i. The database 20-i may be, for example, a large centralized database system, or a memory of a desktop computer, a portable personal computer, a personal digital assistant (PDA) or other processing device, depending upon the nature of the corresponding physical site 12-i.

Detailed Description Text (4):
At least a subset of the processors 18-i each run a local database management system such as that described in A. Silberschatz, H. F. Korth and S. Sudarshan, "Database System Concepts," 3rd edition, McGraw-Hill, 1997, which is incorporated by reference herein. The execution of transactions in local database management systems is further described in J. Gray and A. Reuter, "Transaction Processing: Concepts and Techniques," Morgan-Kaufmann, San Mateo, Calif., 1993, which is incorporated by reference herein. The processors 18-i can be implemented as personal, micro or mainframe computers, workstations, microprocessors, central processing units, application-specific integrated circuits (ASICs) or other digital data processors, as well as various portions or combinations thereof. The databases 20-i may utilize electronic, magnetic or optical storage media, or various combinations thereof, and as previously noted may represent storage locations of data warehouses or data marts having storage capacities of up to a few terabytes, or memories of desktop or portable computers. The storage and retrieval of data items from a database 20-i of a given site 12-i is controlled by the corresponding processor 18-i using the local database management system such that serializability and other desirable properties are maintained. The physical sites 12-i of system 10 may each run multiple transactions, such as read and write transactions, during which data items stored at other sites are accessed and processed. As noted previously, such transactions can create consistency and concurrency problems in that a replicated data item processed at one site may be updated by a transaction running at that site while another non-updated replicated version is used by a transaction running at another site.

Detailed Description Text (9):
As described in the above-cited U.S. patent application of Yuri Breitbart and Henry F. Korth, each physical site of a distributed database system such as system 10 may have a dynamically changing set of virtual sites associated with it, and a concurrency-control protocol may be used to provide global transaction management over the set of virtual sites. Local transaction

management within a given virtual site is provided by the local database management system (DBMS) running at the physical site containing the given virtual site. The replication management techniques of the invention may be part of an integrated system and can therefore utilize the transaction management information from the local DBMSs. This is a significant advantage of the invention over conventional multidatabase systems such as those described in Y. Breitbart, H. Garcia-Molina and A. Silberschatz, "Overview of Multidatabase Transaction Management," VLDB Journal, 1(2), 1992.

Detailed Description Text (11):
A transaction T.sub.i is therefore permitted to be in the completed state even if it did not finish updating of all replicas of data items. As long as it can be ensured that no violation of global serializability could occur as a result of some future operations of local or global transactions, it is unnecessary to wait for global transactions to commit everywhere to satisfy the first condition of completeness. The present invention thus extends the notion of a completed state in order to extend significantly the class of globally serializable schedules which can be generated, while at the same time improving the transaction throughput. If a global transaction is in the committed state, then it may have performed some of its updates at secondary copy sites. Every transaction eventually enters either the committed or the aborted state. However, even if a transaction has committed at all sites, it does not necessarily enter into the completed state.

Detailed Description Text (28):
If it is determined in step 32 that no cycle will be created in the replication graph, T.sub.i is allowed to proceed with execution at its origination site, as shown in step 36. In step 38, a determination is made as to whether T.sub.i has submitted a write operation at a physical site other than its origination site. If it has, the timestamp of T.sub.i is compared in step 40 to the timestamp of the transaction executing the last write operation on the same data item. If the T.sub.i timestamp is less than that of the transaction with the last write operation on the data item, step 42 indicates that the write operation is not performed, and the process returns to step 38 to analyze subsequent operations of T.sub.i. The write operation is thus not performed if another transaction starting after T.sub.i has already executed a write to the same data item. If the T.sub.i timestamp is not less than that of the transaction with the last write operation on the data item, the write operation is sent in step 44 to the local database management system of the appropriate site for execution. A determination is then made in step 46 as to whether T.sub.i has completed. If it is determined in step 38 that T.sub.i has not submitted a write operation at a site other than its origination site, the process moves directly to step 46, bypassing steps 40 and 44. If it is determined in step 46 that T.sub.i has completed, T.sub.i and any of its associated edges are removed from the replication graph, and the process ends for transaction T.sub.i. If T.sub.i has not completed in step 46, the process returns to step 38 to analyze subsequent operations of T.sub.i.

Detailed Description Text (42):
An implementation with distributed graph maintenance will now be considered. The distributed graph maintenance will be described in the context of the COP and MOP protocols, in which graph updates occur when a transaction is about to commit at its origination site. The extension to the SGS protocol is straightforward. In order to synchronize graph updates, each transaction that is ready to commit is assigned a global timestamp, which is broadcast to all physical sites. This timestamp specifies the order in which transactions enter the committed state and allows all sites to agree as to the next transaction to attempt to commit. When a transaction attempts to commit, it tests locally for a cycle in the replication graph and then broadcasts its timestamp along with the decision, either commit or abort. If the decision is to commit, the changes to the replication graph are included in the decision message. This technique depends on the availability of a cheap broadcast mechanism similar to that found in a local area network. There are two broadcasts per transaction, each of which is a long message. The communication overhead for this implementation is 2ntl. Without broadcast, updates are passed from site to site, with each site incorporating its own changes into the message. In either case, the technique incurs the overhead of sending short messages regarding the completion of transactions at secondary sites.

Detailed Description Text (47):

Disconnection of a single physical site, as in the case of a mobile computer disconnecting from a network, is an example of a type of network partition which the protocols of the invention can tolerate easily. If a disconnected site does not contain a primary copy for any data item, then only read transactions can originate at the disconnected site. Since the site is disconnected, none of its data items can be updated. Thus, each read transaction will read a consistent, though possibly not recent, secondary copy. To ensure data availability, the latest committed version of the secondary copies of the data should be copied from the primary sites prior to disconnection. A timestamping technique may be used to avoid unnecessary copying of data. If a disconnected site does contain one or more primary copies of data items, the disconnection could be treated by transactions originating at other sites in a manner similar to that described above for primary data item failure. At the disconnected site, transactions may be allowed to proceed based on the latest copy of the replication graph available to the site. However, none of the transactions originating at the disconnected site is allowed to commit until the site reconnects. After the connection is restored, the replication or copy graph of the disconnected site is merged with the current graph. If the resulting graph is acyclic, each of the transactions from the formerly disconnected site is committed. Otherwise, transactions that introduce a cycle in the graph are aborted and the formerly disconnected site is notified.

Other Reference Publication (12):
T. Minoura, "A New Concurrency Control Algorithm for Distributed Database Systems," Proceedings of the Fourth Berkeley Workshop on Distributed and Computer Networks, pp. 221-234, Aug. 1979.

Other Reference Publication (16):
E. Gelenber and K. Sevcik, "Analysis of Update Synchronization for Multiple Copy Databases," Proceedings of the Third Berkeley Workshop on Distributed Databases and Computer Networks, pp. 69-90, Aug. 1978.

CLAIMS:

11. An apparatus for use in a distributed database system for storing a plurality of data items and controlling access to the data items at a plurality of physical sites configured to communicate over a network, comprising:

at least one processor for managing a plurality of transactions involving the data items, each of the transactions originating at one of the physical sites, wherein the processor is operative to specify sets of data items such that a given set S.sub.i.sup.j of data items at a particular point in time includes replicated data items at a physical site s.sub.j that a given transaction T.sub.i has accessed from an initial operation up to the point in time; to maintain a representation of relationships between the transactions and the sets of data items; and to permit the transaction T.sub.i to enter a completed state even if T.sub.i has not finished updating all replicated data items in the sets of data items associated with T.sub.i, wherein T.sub.i and any sets of data items S.sub.i.sup.j associated therewith are removed from the representation when T.sub.i enters the completed state, such that consistency among the replicated data items is maintained.

Previous Doc Next Doc Go to Doc#

L4: Entry 4 of 9 File: USPT Jul 20, 1999

DOCUMENT-IDENTIFIER: US 5926816 A
TITLE: Database Synchronizer

Abstract Text (1):
A database synchronizer facilitates computing systems which have client-side and server-side applications that share data in similar database structures, but which do not maintain a continuous connection to a single shared data source. In general, a database synchronizer is used to share data among many nodes on the computing system. The database synchronizer is used to synchronize the data in a central database for a particular client with the data on that client's intermittently-connected computer. Updates performed by either client or server are propagated to the other side when a connection is established and eventually from the server to other clients in the system.

Brief Summary Text (10):
A database synchronizer in accordance with the invention facilitates computing systems which have client-side and server-side applications that share data in similar organizational structures, but which do not maintain a continuous connection to a single shared data source. The database synchronizer is a general purpose system which accommodates heterogeneous computers and databases. In general, a database synchronizer is used to share data among many nodes on the computing system. While a central (server) database includes information from all the clients, each remote (client) database is generally limited to data related to the respective client. The database synchronizer is used to synchronize the data in the central database with the data on each client's computer.

Brief Summary Text (12):
A preferred computing system embodying the invention includes a server computer having a central database for storing data therein and any number of client computers having a remote database which includes data replicated from the central database. Both the remote database and the central database organize the data as any number of collections of data with the data being representable as row and columns. In a preferred embodiment of the invention, the databases are relational databases which organize data in tables of rows and columns of data fields. A common structure of shared columns between the server and the client is defined by an aspect of the invention called a table correspondence. A table correspondence is defined as an ordered list of the shared columns. One or more table correspondences are stored in a catalog, another aspect of the invention. Copies of the catalog are stored at the server and client.

Brief Summary Text (13):
A database synchronizer divided between at least one client and a server is used to synchronize the central database and the remote database at an arbitrary time selected by each client. The database synchronizer uses the table correspondence as a common reference between the client and server to identify the tables and columns of the databases which it is to synchronize.

Drawing Description Text (10):
FIGS. 8A and 8B are schematic diagrams illustrating an exemplary client database and central database, respectively.

Detailed Description Text (3):
FIG. 1 is a schematic block diagram of a client-server database system in accordance with the invention. As illustrated, there is a server node 10 and a plurality of client nodes 20a, . . . ,20x, . . . ,20z, each of which having a unique node identifier a, . . . ,x, . . . ,z. The

server node 10 includes at least one processing core 11 comprising a processor and memory for accessing and maintaining a central database 12. Preferably, the central database 12 is a relational database such as, for example, Oracle 7 available from Oracle Corporation.

Detailed Description Text (4):
Each client node 20a, . . . ,20x, . . . ,20z can be a desktop or portable computer, each having at least one processing core 21a, . . . ,21x, . . . ,21z which can include a processor and memory. Each client node 20a, . . . ,20x, . . . ,20z accesses and maintains a respective local replicated database 22a, . . . ,22x, . . . ,22z, each of which is replicated from the central database 12. As such, each of the client local databases 22a,. . . ,22x,. . . ,22z corresponds to a respective subset 12a, . . . , 12x, . . . , 12z of the central database 12.

Detailed Description Text (5):
As illustrated, server database subsets 12a, 12x can overlap so that an individual data element in the server database 12 can be present at a plurality of client databases 22a, 22x. Each of the client local databases 22a, . . . ,22x, . . . ,22z can also include additional data fields which are not related to data fields in the central database 12.

Detailed Description Text (7):
The server node 10 can preferably store data for all clients and support multiple simultaneous users, e.g., a mainframe computer or computer cluster. The client nodes 20 are preferably autonomous personal computers, such as laptop or hand-held computers, which intermittently communicate with the server node 10 through a communications network 5, such as the telephone network. In general, although not required, the databases on the server and clients are heterogeneous. As such the server cannot generally control the database semantics and operations at the clients. Furthermore, the server cannot rely on a knowledge of the database logic at the clients.

Detailed Description Text (8):
As illustrated, each of the client nodes 20a, . . . ,20x, . . . ,20z can establish a respective communication link 25a,. . . ,25x, . . . ,25z with the communications network 5, which is linked to the server node 10 through a communications link 15. The communications links 15, 25 can be analog or digital links. To that end, each node includes a respective data transceiver 13, 23 such as an analog modem, a cellular modem, a digital (e.g., ISDN) modem, or an infrared (IR) transceiver.

Detailed Description Text (9):
Over time, the central database 12 can be modified by users to insert, update and delete rows, columns and data fields. These modifications to the central database 12 can be accomplished by users at the server or by users at one or more of the client nodes 20. Similarly, a user at a particular client node 20x can also modify the client local database 22x over time by inserting, updating and deleting data fields. Because the client nodes 20 are typically disconnected from the server node 10, corresponding data fields in the client databases 22 and the central database 12 tend to diverge over time.

Detailed Description Text (11):
The divergent data makes it difficult for the clients and server to share data. That problem can be addressed by an ad hoc solution on the client-server network. Such solutions, however, are time consuming to develop, difficult to master, nongeneral in purpose, and suffer from various reliability problems due to poor error recovery.

Detailed Description Text (14):
A preferred embodiment of the invention provides for a synchronization system which brings the client local database 22x into synchronization with the central database 12. To facilitate the synchronization, a database synchronizer 17, 27 comprising programming instructions and data is resident on the server node 10 and client nodes 20, respectively. During the synchronization process, database modifications are propagated in both directions and confficts are detected and resolved so that data can be shared among a plurality of nodes. The synchronization does not, in general, cause corresponding tables to become identical, because not all columns of the corresponding tables are replicated and some rows may be excluded from the synchronization

process by filters.

Detailed Description Text (17):
FIG. 2 is a schematic block diagram of a client node 20x having a client-side database synchronizer embodying the invention. As illustrated in FIG. 1, the client node 20x includes a processing core 21x, a local database 22x and a client-side database synchronizer 27x. The local database 22x includes a plurality of data tables 22x-1, . . . ,22x-X. Although three tables are illustrated, it will be understood that the local database 22x can include as few as one table or many more tables. The client-side database synchronizer 27x comprises a client catalog structure 60x containing table correspondences 60x-1, . . . ,60x-X and before-image logs 62x-1, . . . , 62x-X for each table 22x-1, . . . ,22x-X. Each table correspondence 60x-1, . . . ,60x-X includes an ordered, sequential listing of all of the replicated columns in each of the tables 22x-1, . . . ,22x-X of the local database 22x. Each before-image log 62x-1, . . . ,62x-X corresponds to the last synchronized values of the replicated columns in respective tables 22x-1, . . . ,22x-X, of the local database 22x.

Detailed Description Text (18):
FIG. 3 is a schematic block diagram of a server having a server-side database synchronizer 17 embodying the invention. As described above, the server node 10 includes a processing core 11 and a central database 12. In addition to communicating with the central database 12, the processor 11 communicates with the server-side database synchronizer 17.

Detailed Description Text (19):
The server-side database synchronizer 17 includes a server catalog structure 70, a server update log 82, and a last confirmed refresh table 84. The central database 12 is divided into replicated tables 12a, . . . ,12x, . . . ,12z. The server catalog structure 70 includes a table correspondence 72a, 72x, 72z for each replicated table 12a, . . . ,12x, . . . ,12z. The server catalog structure 70 also includes entries 75a, . . . ,75x, . . . ,75z corresponding to, and having information about, client nodes 20a, . . . ,20x, . . . ,20z. There is one server update log (SUL) 82a, 82x, 82z per table replicated from the central database 12. There is preferably one refresh table 84 for all data tables 12a, 12x, 12z.

Detailed Description Text (39):
In a preferred embodiment of the invention, the server node 10 weighs in favor of either the current value in the central database 12 or the updated value from the updating client node 20x. For specific data fields, the server node 10 can resolve database conflicts in favor of some client nodes but not other client nodes. Further details of the conflict resolution process are described below.

Detailed Description Text (40):
If the conflict is resolved in favor of the client (step 225) or if there is no conflict, then at step 230, the server node 10 modifies data in the central database 12, and a record of the client's operation, including the client's numeric identifier, is added to the server update log at step 235. For additional rows (step 240), processing returns to step 210.

Detailed Description Text (55):
As part of each refresh pass, the server sends a checksum for its replica of the data. The client saves the checksum value and uses it the next time it is scanning the before-image table (for a refresh or propagate updates operation). If the checksums do not match, the server is notified, the modification messages generated during the first step are discarded, and the client is marked and disabled to await intervention. Thus, an extra pass over the client data is avoided.

Detailed Description Text (82):
FIGS. 8A and 8B are schematic diagrams illustrating table 12a in a central database 12 and an exemplary table 22x-a in a client database 22x, respectively. The central database 12 and the client database 22x include tables organized in rows and columns of data fields, which are initially synchronized as shown. The values of the non-key data fields are illustrated by reference characters A-V. Also shown are the key columns K.sub.12, K.sub.22 and the unique key values stored therein.

Detailed Description Text (83):
As illustrated, the first row R1.sub.22 of the client table 22x-a is a replicated subset of the first row R1.sub.12 of the central database table 12a. The second row R2.sub.22 of the client table 22x-a is a replicated. subset of the fourth row R4.sub.12 of the central database table 12a. Only the first, second and fourth columns C1.sub.12, C2.sub.12, C4.sub.12 of the central database table 12a are replicated to the columns C1.sub.22, C2.sub.22, C3.sub.22 of the client table 22x-a. The client table 22x-a also includes a fourth column C4.sub.22 which does not correspond to any column in the central database table 12a and a row R3.sub.22 which does not correspond to any row in the central database table 12a.

Detailed Description Text (85):
FIGS. 9A and 9B are schematic diagrams of a table view of a replicated server-side and client-side database table, respectively. As illustrated, the server table view Ts is a filtered subset of the central database table 12a. Likewise, the client table view Tc is a filtered subset of the client database table 22x-a. The table correspondence describes which columns of tables 12a and 22x-a appear in these views, and the filters in the table correspondence provide predicates that must be satisfied for rows to appear in the views.

Detailed Description Text (98):
As illustrated, there is one conflict in the databases, which is detected by the server 10 upon receipt of the update messages from client 20x. Specifically, the change to the data field R(1) C1 is in conflict with the change previously made by client 20a. The conflict is resolved at the server 10 and the central database table 12a is updated.

Detailed Description Text (145):
Although the invention has been described with reference to relational database tables, other database models can also be used at either the server or the client. For example, both object-oriented and relational databases structure data as a plurality of sub-collections of objects (or rows), each of which is of the same type (or has the same fields). A table of rows can thus be modeled as a collection of objects (often called a "class") with like attributes. Conversely, data in an object-oriented database may be representable as tabular data having rows and columns of data fields. Consequently, the invention also applies to object-oriented databases to the extent that a given class has properties equivalent to that of a table in a relational database (e.g., data modifiable using insert, update and delete operations). Similarly, the invention can be applied to other database models including, but not limited to, hierarchial and network (CODASYL) databases.

CLAIMS:

1. A method of synchronizing values of data items on a plurality of computers, comprising the steps of:

storing a data item having a value at a first computer;

at the first computer, maintaining a log of modification operations to the value of the data item, the value of the data item being modifiable by the first computer and in response to actions at a plurality of computers, including at a second computer;

at the first computer, deriving a single effective operation from a plurality of modification operations maintained in the log, the effective operation yielding the same result on the value of the data item as would the plurality of modification operations; and

using the effective operation to synchronize the value of the data item at the first computer with a value of a replica data item stored at the second computer.

4. The method of claim 1 further comprising the step of detecting a conflict between modifications to the value of the data item and the value of the replica data item using the effective operation.

5. The method of claim 4 wherein the step of detecting comprises:

determining a deduced modification operation to propagate the value of the replica data item from the second computer to the first computer; and

comparing the effective operation with the deduced modification operation to yield a conflict state.

7. A system for synchronizing values of data items on a plurality of computers, comprising:

a value of a data item stored at a first computer;

a log of modification operations to the value of the data item maintained at the first computer, the value of the data item being modifiable by the first computer and in response to actions at a plurality of computers, including at a second computer;

an effective operation derived at the first computer from a plurality of modification operations in the log, the effective operation calculated to yield the same result to the value of the data item as would the plurality of modification operations; and

a database synchronizer to synchronize the value of the data item with a value of a replica data item stored on the second computer using the effective operation.

10. The system of claim 7 wherein the database synchronizer comprises a conflict detector to detect a conflict between modifications to the value of the data item and the value of the replica data item using the effective operation.

11. The system of claim 10 wherein the conflict detector comprises:

a deduced modification operation calculated to propagate the value of the replica data item from the second computer to the first computer; and

a conflict state yielded from comparing the effective operation with the deduced modification operation.

Previous Doc Next Doc Go to Doc#